



Kprobes and Systemtap Status

Sony India Software Centre
Sony Corporation

Madhvesh Sulibhavi (madhvesh.s@ap.sony.com)

April 7th, 2009



Contents

- **Overview**
- **Kprobes MIPS Arch details**
- **Systemtap for MIPS Arch Details**
- **Quick look at Systemtap GUI for Embedded Systems**
- **Kprobes Support for PPC32 (BookE) Mainline Status**
- **References**



Overview (1)

- This presentation covers the work done internally within Sony with respect to kprobes, jprobes and kretprobes for architectures like MIPS, PPC and ARM.
- Also this presentation highlights on systemtap status for MIPS arch and challenges involved in porting systemtap to new architecture.
- Finally a quick look at Systemtap GUI project for embedded systems like MIPS target.



Overview (2)

- **Prior Work Update**

- Sony had implemented Kprobes for ARM, MIPS and PPC architectures using kernel version 2.6.16.38 way back in 2007. This was presented during ELC-07 event here.
- This support included only kprobes, but no jprobes and kretprobes features.
- Recently we could port the previous versions of kprobes to kernel versions like 2.6.23 and 2.6.29 along with new features like Jprobes and Kretprobes support.



Overview (3)

- **Prior Work Update**

- Systemtap user space tool is integrated with these kprobes for architectures like MIPS and PPC (BookE) versions for both kernel versions 2.6.23 and 2.6.29.
- Systemtap GUI project is modified to use with cross tools and embedded systems.



Kprobes Quick Overview

- In the Linux kernel, there are currently three types of probes:
 - kprobes, jprobes, and kretprobes (also called return probes).
- A kprobe can be inserted on virtually any instruction in the kernel.
- A jprobe is inserted at the entry to a kernel function, and provides convenient access to the function's arguments.
- A Kretprobe is the one which fires when a specified function returns.
- Refer Documentation/Kprobes.txt in kernel source tree



Kprobes and Systemtap for MIPS



MIPS Target Environment Details

- **Target Board**
 - Toshiba RBHMA4400 (TX4937)
- **OS**
 - Linux 2.6.23.17
 - Linux 2.6.29
- **Configuration**
 - CONFIG_KPROBES=y
 - CONFIG_KRETPROBES=y
 - CONFIG_HAVE_KPROBES=y
 - CONFIG_HAVE_KRETPROBE S=y
 - CONFIG_DEBUG_FS=y



Hardware Features:

TX4937 @ 300Mhz.

System Clock @ 66Mhz

USB 1.1 Host Controller



Status of Kprobes for MIPS Arch(1)

- Recently Sony has forward ported the earlier implemented 2.6.16.38 kernel kprobes patches to 2.6.23 and now for 2.6.29 kernel.
- Design Points
 - **MIPS arch contains “break” instruction using which break point can be achieved.**
 - **MIPS does not have hardware single step support and hence single step feature is implemented in software using “break” instruction with different code value**
- The other design is same as explained during ELC-07 slides (refer the slides from 10 to 16)



Status of Kprobes for MIPS Arch(2)

- **Major changes to recent 2.6.29 kernel Kprobes since 2.6.16.39 include**
 - Arch specific die notifier handling code is removed and refers to common die notifier code mechanism.
 - Kretprobe code is simplified to make it use common code for arch_prepare_kretprobe
 - Support for -rt Kernel (locks are replaced)
 - Kprobes on/off using debugfs support



Status of Kprobes for MIPS Arch(3)

- **Major changes (Continued)**
 - Support for kretprobe blacklist
 - kretprobe blacklists will prohibit users from inserting return probes on the function in which kprobes can be inserted but not kretprobes.
 - Earlier only one kretprobe handler used to execute at a time. This is improved now using fine-grain locks which perform better on SMP systems.



MIPS Kprobes Limitations

- Not tested under SMP configuration
- Following instruction types are not supported
 - Jump register or jump and link register (ex: jr, jalr)
 - Jump And Link and Jump (ex: jal, j)
 - Branch instructions (ex: bltz, bgez, beq, bne, blez, bgtz)
 - coprocessor instructions (ex: cop1)



Systemtap for MIPS Arch



Systemtap Overview (1)

- Systemtap is a tool that allows anyone to deeply investigate the behavior of the kernel and even user space applications in order to discover error conditions, performance issues, or just to understand how the system works
- The goal of systemtap project is to provide debugger-quality visibility into entire running system, ideally any variable at any statement in any thread !!



Systemtap Overview (2)

- Run fast, non-intrusively, to minimize probe effect and gross disruption
- Refer more details of systemtap from its wiki
 - <http://sourceware.org/systemtap/wiki>



Systemtap Overview (3)

- Systemtap cannot be used as it is, to build using cross tools on host system. It requires some changes to its configure system
- There are two ways to work with systemtap.
 - Cross Compilation environment
 - Native compilation environment



Systemtap Overview (4)

- **Cross Compilation Environment**
 - It is fast
 - Best suited for embedded systems
 - But test procedure needs to be fitted for host/target environment
- **Native compilation Environment**
 - One need not to worry about header, library paths
 - Can be tested immediately without any changes



Systemtap Overview (5)

- Drawbacks of Native Compilation
 - It isn't fast enough
 - Most of the applications for embedded systems are configured for Cross-Compilation!
 - Not many choices in compilers
 - Native compiler faces different problems
- Sony explored on both approaches and identified the changes required for systemtap to work using cross tools.



Systemtap for MIPS Arch (1)

- **MIPS Arch support include following changes to Systemtap**
 - Support for cross compiler build system (configure related changes and others)
 - Supporting run time arch specific registers definitions and functions
 - Defining arch specific low level definitions for put_user and get_user
 - gcc defined arch specific long long definitions (from libgcc)



Systemtap for MIPS Arch (2)

- **MIPS Systemtap Changes (contd)**
 - Arch specific stack tracing functions
 - Systemtap defined tapset functions some of which are specific to arch defined syscalls
 - Bypassing the syscalls which are x86 specific like `sys_get_thread_area` and `sys_iopl`. This change looks like temporary as of now and will be made as MIPS arch specific.



Systemtap for MIPS Arch (3)

- Using the latest systemtap release 0.9.5, Sony has ported the MIPS systemtap patches to this version and able to use it with 2.6.29 kernel
- Key benefits noticed while using Systemtap-0.9.5
 - Kernel tracepoints are now supported for probing predefined kernel events **without any debuginfo symbols presence on the target system.**
 - Tracepoints incur less overhead than kprobes, and context parameters are available with full type information
 - Users can prevent interrupt reentrancy in their script (by using `-DINTERRUPTIBLE=0`)



Systemtap for MIPS Arch (4)

- Systemtap-0.9.5 Files modified for MIPS support
 - `src/buildrun.cxx`
 - `src/configure`
 - `src/tapsets.cxx`
 - `src/tapset/i686/syscalls.stp`
 - `src/runtime/arith.c`
 - `src/runtime/arith.c`
 - `src/runtime/copy.c`
 - `src/runtime/regs.c`
 - `src/runtime/regs.h`
 - `src/runtime/stack-mips.c`
 - `src/runtime/stack.c`
 - `src/tapset/errno.stp`
 - `src/runtime/loc2c-runtime.h`
 - `src/tapset/syscalls.stp`
 - `src/tapset/nd_syscalls.stp`
 - `src/tapset/aux_syscalls.stp`



Preparing/Using Systemtap in Cross Environment



Systemtap Preparation (1)

- **Using Native environment**
 - This is not explained here as plenty of documentation is available in [systemtap wiki page](#).
- **Using cross tool environment**
 - This involves preparing the systemtap for **host environment**
 - and preparing for **target environment**



Systemtap Preparation (2)

- **Preparing systemtap using cross tools for host system**
 - The systemtap is applied with MIPS arch specific systemtap prepared patches relevant for host system build
 - Export the CC and CPP variables to cross compiler path (ex: CC=/usr/local/xxx/mips-linux-gcc)
 - Configure the systemtap. Please note you need to specify elfutils source path also
Ex: ./configure --with-elfutils=/root/elfutils-0.131
 - Then install using 'make install'



Systemtap Preparation (3)

- **Preparing systemtap using cross tools for host system (Contd)**
 - The installation step will install the following in the cross tools host system path
 - /usr/local/<cross-tool-path>/bin/stap
 - This is main executable which will be used for systemtap program compilation
 - /usr/local/<cross-tool-path>/lib/systemtap
 - This folder will be created with all shared libraries
 - /usr/local/<cross-tool-path>/share/systemtap/runtime
 - This folder gets created with all systemtap runtime files
 - /usr/local/<cross-tool-path>/share/systemtap/tapset
 - This folder gets created with all systemtap builtin tapsets



Systemtap Preparation (4)

- **Preparing systemtap using cross tools for target system**
 - The systemtap is applied with MIPS arch specific systemtap prepared patches **relevant for target system build**.
 - Export the CC and CPP variables to cross compiler path (ex: CC=/usr/local/xxx/mips-linux-gcc)
 - Configure the systemtap and do 'make install'



Systemtap Preparation (5)

- **Preparing systemtap using cross tools for target system (Contd)**
 - The installation step will install the following and they need to be used in the target root file system
 - /<root-file-system>/bin/staprun
 - This is main executable on the target which need to be used to execute systemtap kernel modules prepared on host system
 - /<root-file-system>/libexec/systemtap/stapio
 - This is required to use the staprun on target side.



Systemtap example (1)

- **Building systemtap on host system**
- Below sample program does probing for do_gettimeofday kernel function

stap_gtod.stp

```
#!/usr/bin/env stap
  probe kernel.function("do_gettimeofday") { }
  probe kernel.function("do_gettimeofday").return
  { }
```



Systemtap example (2)

- Build the kernel with Kprobes+Systemtap related config options. (CONFIG_KPROBES, CONFIG_KRETPROBES, CONFIG_DEBUG_INFO, CONFIG_DEBUG, CONFIG_RELAY, CONFIG_DEBUG_FS)
- **Prepare the systemtap kernel module on host system**
 - `/usr/local/mips-sony-linux/devel/bin/stap -r 2.6.29 -m mips_stap_gtod -p4 stap_gtod.stp`
 - The above step will generate the kernel module by name `mips_stap_gtod.ko`
- **Executing the systemtap kernel module on target**
 - Copy the kernel module built on host to target system and the execute
 - `/bin/staprun mips_stap_gtod.ko`
 - The above command will run the program and output the result.



Kprobes and Systemtap Overhead Measurement Data



Kprobes and Systemtap Overhead

- Sony has measured the kprobes and systemtap overhead for 2.6.23.17 and 2.6.29 kernel.org kernels for TX49 target MIPS arch
- Environment Details
 - Linux Kernel 2.6.23.17 and 2.6.29
 - Binutils-2.17.50
 - Gcc-4.1.2
 - glibc-2.7
 - TX49 target



Kprobes and Systemtap Overhead(contd)

- Overhead is measured in 3 conditions
 - WITHOUT kprobes config option
 - WITH kprobes config option
 - WITH kprobes + systemtap related config options (CONFIG_RELAY, CONFIG_DEBUG_FS, CONFIG_DEBUG_KERNEL, CONFIG_DEBUG_INFO)



Kprobes and Systemtap Overhead (Contd)

- The benchmark tests were used from earlier Sony released kprobes test modules during 2007 in CELF wiki page.
- Some of the systemtap programs from its test suite are also used for this.
- Kprobe Overhead Measurement Test (k-009.c)
 - This test probes the `do_gettimeofday` function with empty pre and post handlers
- User Space Measurement Test (u-009.c)
 - This test measures the processing time for multiple calls to `gettimeofday()`



Kprobes and Systemtap Overhead (Contd)

- **Systemtap Performance Measurement Test**
 - This program is developed to probe gettimeofday. The test is of just 2 lines unlike the plenty of code as in kprobes (k-009.c)

stap_gtod.stp

```
#!/usr/bin/env stap
probe kernel.function("do_gettimeofday") { }
probe kernel.function("do_gettimeofday").return
{ }
```



Kprobes and Systemtap Overhead (Contd)

- Benchmark Test
 - This benchmark test repeats the `gettimeofday` system call for 10 seconds and counts the number of calls. After that, calculated the average processing time per call.
 - This program looks as below (`runtest.sh`)

```
#!/bin/sh
  for i in `seq 1 10`;
  do
    /root/u-009
    sleep $1
  done
```



Kprobes and Systemtap Overhead (Contd)

- Results by using **Systemtap-v0.8 + 2.6.23.17 MIPS Kprobes Patches**
 - gettimeofday() probe execution overhead details (All measurements in usecs)

Environment	Default (Using only u-009.c and No Kprobes)	KPROBES enabled and USED (k-009.c and u-009.c)	Systemtap + Kprobes enabled and USED (stap_gtod.stp and u-009.c)
2.6.23.17 kernel and Systemtap-v0.8	1.173863	3.306430	6.392914



Kprobes and Systemtap Overhead (Contd)

- Results by using **Systemtap-v0.9.5 + 2.6.29** with MIPS Kprobes Patches
 - gettimeofday() probe execution overhead details (All measurements in usecs)

Environment	Default (Using only u-009.c and No Kprobes)	KPROBES enabled and USED (k-009.c and u-009.c)	Systemtap + Kprobes enabled and USED (stap_gtod.stp and u-009.c)
2.6.29 kernel and Systemtap-v0.9.5	1.244693	3.388809	6.565220



Kprobes and Systemtap Overhead (Contd)

- Kprobes when present in the kernel, doesnot have much overhead
- Kprobes overhead is less as compared to systemtap, but systemtap has other advantages from user space



Using Systemtap GUI for Embedded Systems



Using Systemtap GUI

- Overview
- Systemtap GUI Changes for embedded systems
- Sample programs using systemtap GUI.



Systemtap GUI Overview (1)

- Systemtap GUI provides an easy and effective front end for the systemtap tool. It works as a data visualization and analysis tool
- It is an Integrated Development Environment for the systemtap scripting language on host system
- Internally systemtap GUI works on client and server model.



Systemtap GUI Overview (2)

- For embedded systems separating the client and server was required to make them work using cross tools on host system.
- However the systemtap GUI works on the basis of Native environment. It does not support the cross tool environment explained in the earlier slides for systemtap.
- So the target system needs to be installed with debug info symbols and systemtap runtime libraries to work with systemtap GUI!!!



Systemtap GUI Overview (3)

- **Systemtap Client**

- This need to be prepared on the host system where cross tools are installed.
- The systemtap GUI client part requires the Java installation (Sun JRE 1.5 or above) to be installed in the system



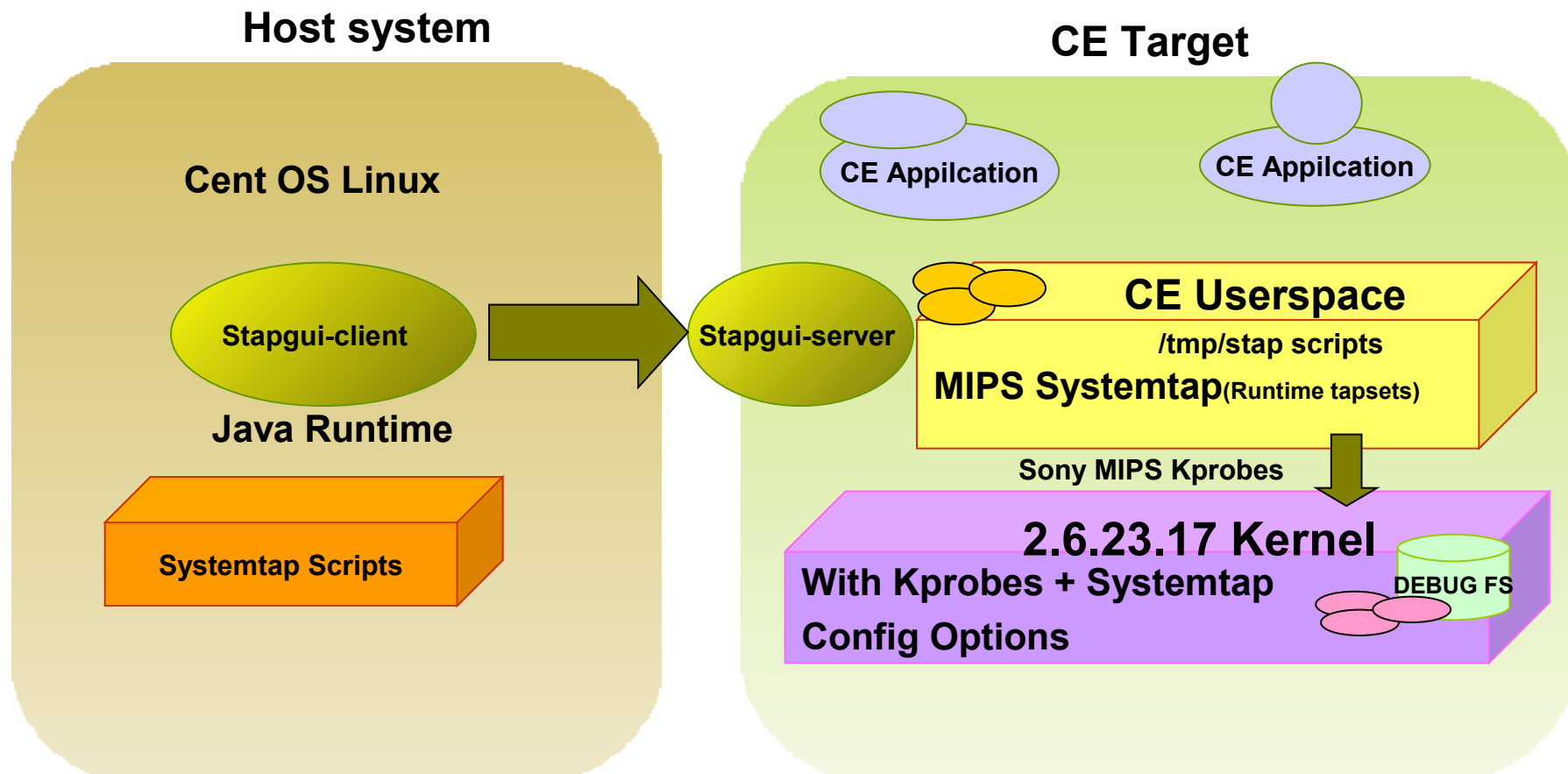
Systemtap GUI Overview (4)

- **Systemtap Server**

- This need to be executed on embedded target system where the stap is installed.



Systemtap GUI Overview (5)





Systemtap GUI Changes (1)

- Following are some of the changes what Sony did to use the systemtap GUI on embedded systems
- The systemtap GUI server part had to be built using cross tools to make it run on embedded target
 - As part of this some of the Makefiles had to be modified to make them cross-compiler friendly
- Systemtap GUI server side file datamanager.cpp was modified to make it work on target side.
- For the systemtap client part, No changes were required.



Systemtap GUI Changes (2)

- But still we have not rectified loading of systemtap scripts from host to target automatically.
- Currently prior to using systemtap GUI on host side, we have placed all systemtap scripts on the target side manually!!
- With the above minimal changes, we could execute the programs from host side.



Systemtap GUI Changes (3)

- But this requires the compiler+systemtap to be installed on the target system. Also kernel should be built on the target and the debug symbols need to be installed



Using programs in Systemtap GUI

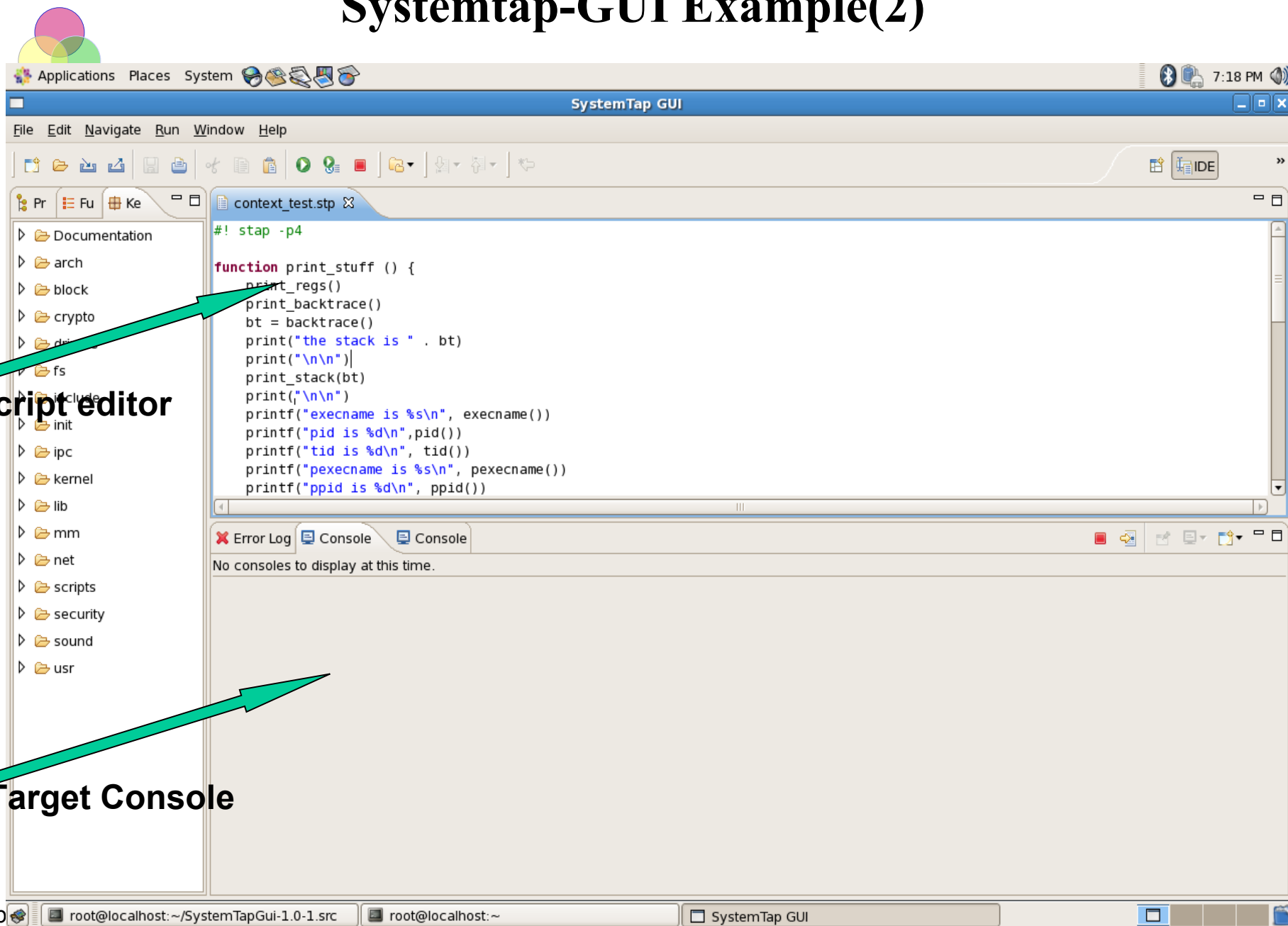
- In Host Machine (Ex: Cent OS 5.1):
 - Start stapgui-client
 - Open the systemtap script
 - Execute the script
- In Target Machine (Ex: MIPS board):
 - Start the stapgui-server



Systemtap GUI Example (1)

- In the systemtap test suite, the following example is referred
- `./testsuite/buildok/context_test.stp`

Systemtap-GUI Example(2)





Systemtap-GUI Example(3)

The screenshot displays the SystemTap GUI interface. The top menu bar includes 'File', 'Edit', 'Navigate', 'Run', 'Window', and 'Help'. The toolbar contains various icons, including a green play button. A red arrow points to this play button with the text 'Click here to run the stap script on target'. The main editor window shows a script named 'context_test.stp' with the following content:

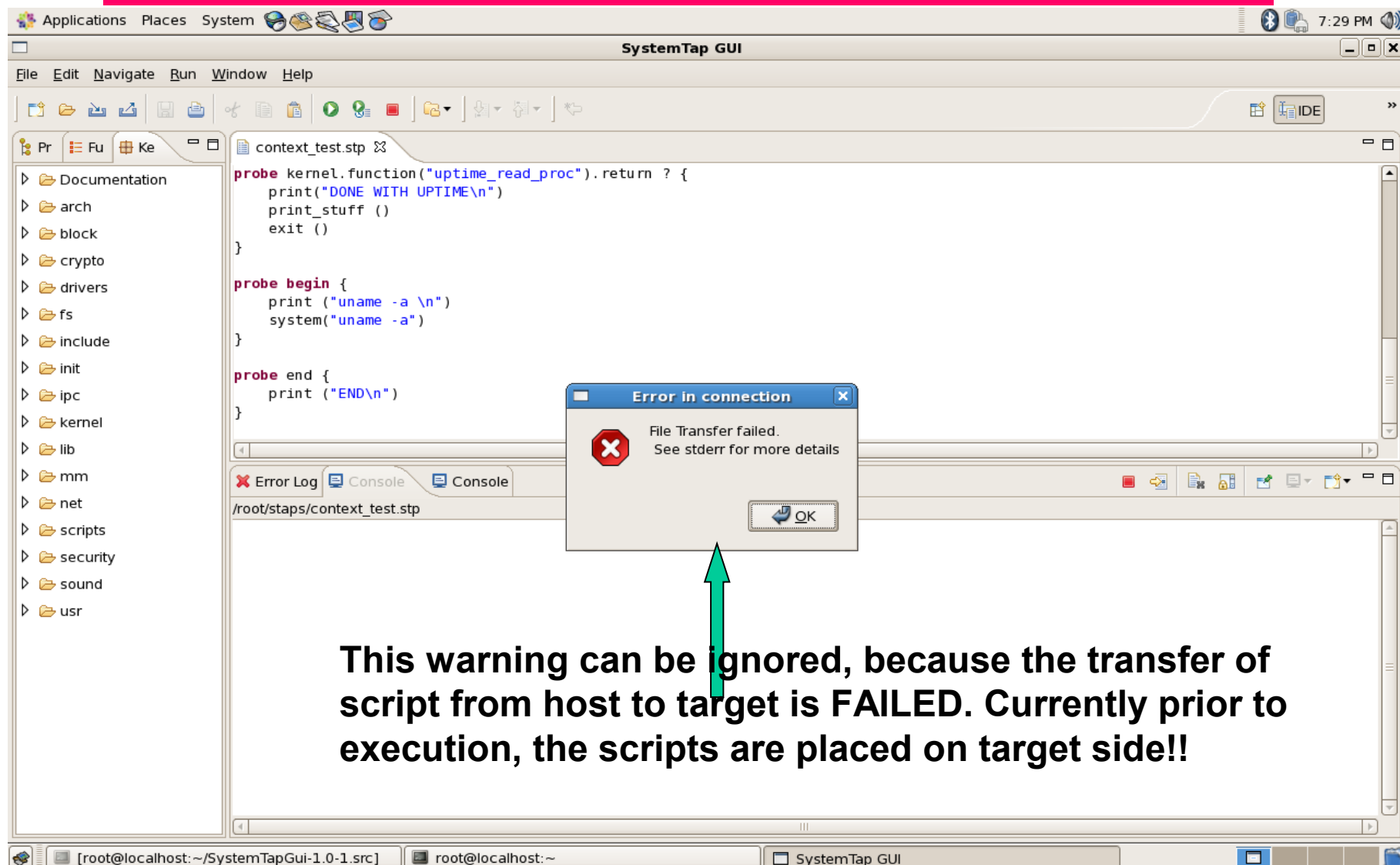
```
#!/bin/sh stap -p4

function print_stuff () {
    print_regs()
    print_backtrace()
    bt = backtrace()
    print("the stack is " . bt)
    print("\n\n")
    print_stack(bt)
    print("\n\n")
    printf("execname is %s\n", execname())
    printf("pid is %d\n", pid())
    printf("tid is %d\n", tid())
    printf("pexecname is %s\n", pexecname())
    printf("ppid is %d\n", ppid())
}
```

Below the editor is a console window with tabs for 'Error Log' and 'Console'. The console output shows the path '/root/staps/context_test.stp'. The bottom status bar indicates the current directory is '/root/staps/context_test.stp' and shows a file named 'context_test-stap.png'.

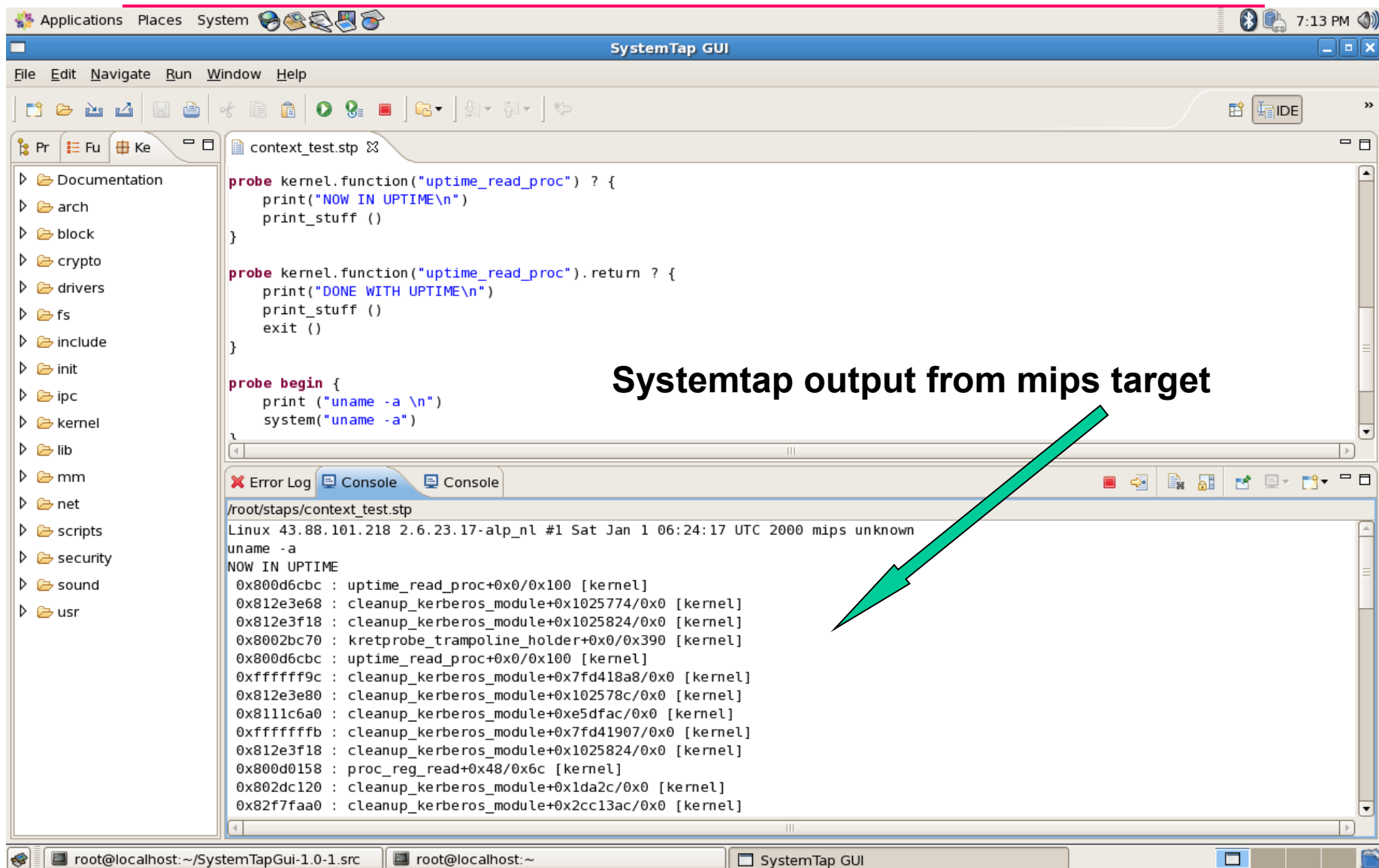


Systemtap-GUI Example(4)





Systemtap-GUI Example(5)



The screenshot shows the SystemTap GUI interface. On the left is a file browser showing a directory tree with folders like Documentation, arch, block, crypto, drivers, fs, include, init, ipc, kernel, lib, mm, net, scripts, security, sound, and usr. The main editor displays a script named `context_test.stp` with the following content:

```

probe kernel.function("uptime_read_proc") ? {
    print("NOW IN UPTIME\n")
    print_stuff ()
}

probe kernel.function("uptime_read_proc").return ? {
    print("DONE WITH UPTIME\n")
    print_stuff ()
    exit ()
}

probe begin {
    print ("uname -a \n")
    system("uname -a")
}
    
```

Below the editor is a console window showing the output of the script execution. A large green arrow points from the text "Systemtap output from mips target" to the console output. The console output is as follows:

```

/root/staps/context_test.stp
Linux 43.88.101.218 2.6.23.17-alp_nl #1 Sat Jan 1 06:24:17 UTC 2000 mips unknown
uname -a
NOW IN UPTIME
0x800d6cbc : uptime_read_proc+0x0/0x100 [kernel]
0x812e3e68 : cleanup_kerberos_module+0x1025774/0x0 [kernel]
0x812e3f18 : cleanup_kerberos_module+0x1025824/0x0 [kernel]
0x8002bc70 : kretprobe_trampoline_holder+0x0/0x390 [kernel]
0x800d6cbc : uptime_read_proc+0x0/0x100 [kernel]
0xffffffffc : cleanup_kerberos_module+0x7fd418a8/0x0 [kernel]
0x812e3e80 : cleanup_kerberos_module+0x102578c/0x0 [kernel]
0x8111c6a0 : cleanup_kerberos_module+0xe5dfac/0x0 [kernel]
0xfffffffffb : cleanup_kerberos_module+0x7fd41907/0x0 [kernel]
0x812e3f18 : cleanup_kerberos_module+0x1025824/0x0 [kernel]
0x800d0158 : proc_reg_read+0x48/0x6c [kernel]
0x802dc120 : cleanup_kerberos_module+0x1da2c/0x0 [kernel]
0x82f7faa0 : cleanup_kerberos_module+0x2cc13ac/0x0 [kernel]
    
```

The bottom status bar shows the current session as `root@localhost: ~/SystemTapGui-1.0-1.src` and the application name as `SystemTap GUI`.



Kprobes PPC32 (BookE) Status



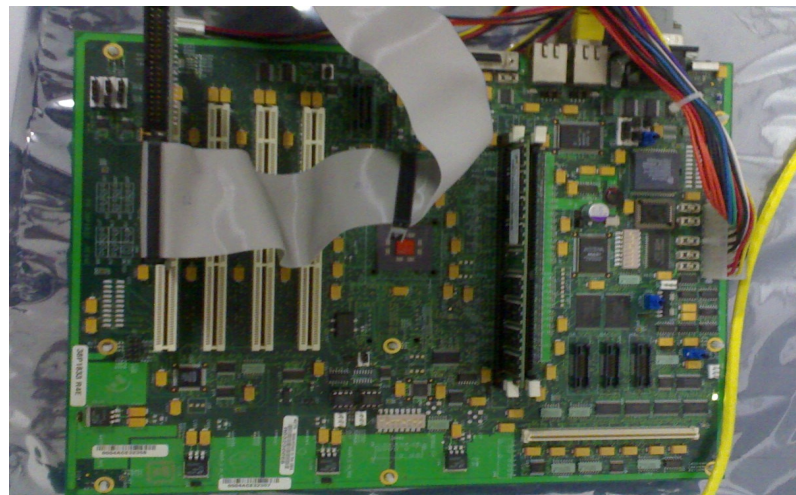
PPC32 Arch Kprobes Status

- Target Environment Details
- Kprobes Status in Mainline Kernel



PPC32 Target Environment Details

- **Target Board**
 - Ebony (PPC440gp)
- **OS**
 - Linux 2.6.23.17
 - Linux 2.6.29
- **Configuration**
 - CONFIG_KPROBES=y
 - CONFIG_KRETPROBES=y
 - CONFIG_HAVE_KPROBES=y
 - CONFIG_HAVE_KRETPROBE S=y
 - CONFIG_DEBUG_KERNEL=y
 - CONFIG_DEBUG_FS=y



Hardware Features:

440GP Processor @ 500 Mhz
System Clock @ 66.66Mhz



PPC32 Kprobes Status in Mainline (1)

- Sony had provided kprobes support for PPC32 BookE versions using 2.6.16.38 kernel. This was presented during ELC-07 event also!
- Last year as part of ppc arch merge to powerpc, the BookE version patches were forward ported during 2.6.26.rc6 kernel version.
- Refer the details in the discussion threads here
 - <http://ozlabs.org/pipermail/linuxppc-dev/2008-June/05>
 - <http://ozlabs.org/pipermail/linuxppc-dev/2008-June/05>



PPC32 Kprobes Status in Mainline (2)

- The major challenge in handling the PPC32-BookE versions single step exception handling is addressed completely now.
- **The mainline kernel > v2.6.27 contains the support for ppc-44x BookE versions kprobes support.**



PPC32 Kprobes Status in Mainline (3)

- **Major Kprobes changes in >v2.6.27 since 2.6.16.38 include**
 - The 2.6.16 version of kprobes was not having jprobes and kretprobes support and now this is supported.
 - Earlier code was having a bug to accept async exception while attempting to single step the probe point. This is fixed now.
 - Kprobes on/off using debugfs support



PPC32 Kprobes Status in Mainline (4)

- **Major Kprobes changes in >v2.6.27 since 2.6.16.38 (contd)**
 - Support for kretprobe blacklist.
 - kretprobe blacklists will prohibit users from inserting return probes on the function in which kprobes can be inserted but not kretprobes.
 - Support for -rt Kernel (locks are replaced)



PPC32 Kprobes Status in Mainline (5)

- So the kprobes support present in kernel > v2.6.27 can be used for ppc-44x based embedded targets also without any changes.



Source Patches

- The latest patches for MIPS kprobes and systemtap will be released in the systemtap mail list very soon!
- However the patches for 2.6.16.38 are available from below link
 - <http://tree.celinuxforum.org/CelfPubWiki/PatchArchive>



References

- <http://sourceware.org/systemtap/wiki>
- <http://sourceware.org/systemtap/tutorial/>
- http://sourceware.org/systemtap//SystemTap_Beginners_Gui